Score-based Generative Models for Calorimeter Shower Simulation

Vinicius M. Mikuni, Ben Nachman

Detector simulation

- Calorimeters **expensive to simulate**:
  - Full detector simulation of a particle can take up to **a minute** and we still need **billions of particles simulated**
- For previous LHC runs, detector simulation used around **40% of all computing resources** and may go beyond the available budget for future runs

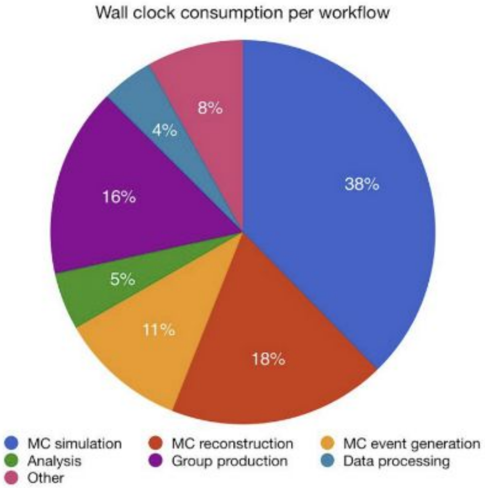
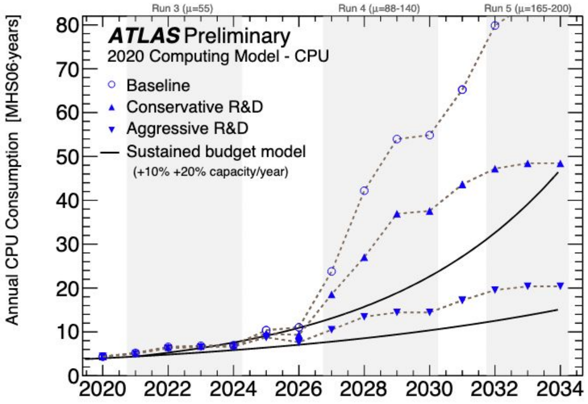
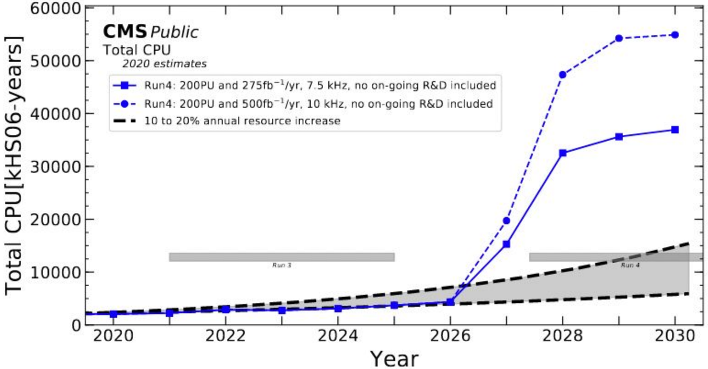


Figure 1: ATLAS CPU hours used by various activities in 2018





Diffusion models

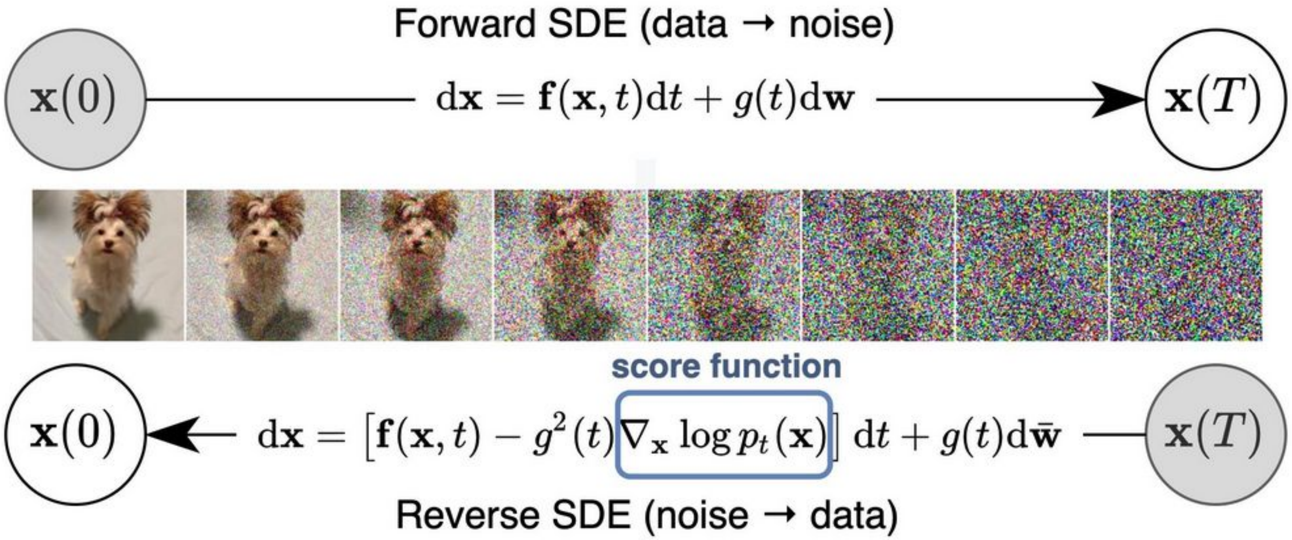
"An astronaut lounging in a tropical resort in space in a photorealistic style"

https://openai.com/dall-e-2/

Diffusion models

More generally, we can define a **forward diffusion process** that slowly corrupts our input data over time

- **Reversing** the diffusion process is the same as a generative model!
- With **f** and **g** fixed, the goal is to estimate the **score function**, or the **gradient of the log probability distribution**



Score-matching

- The **breakthrough** insight was to notice that approximating the score function of the data is **equivalent** to **approximating the score function of a smearing function** that is used to perturb the data, **minimizing**:

$$\frac{1}{2}\mathbb{E}_t\mathbb{E}_{p_t(\tilde{x})}\lambda(t)\left[\|s_\theta(\tilde{x},t) - \nabla_{\tilde{x}}\log p_t(\tilde{x}|x_0)\|_2^2\right]$$

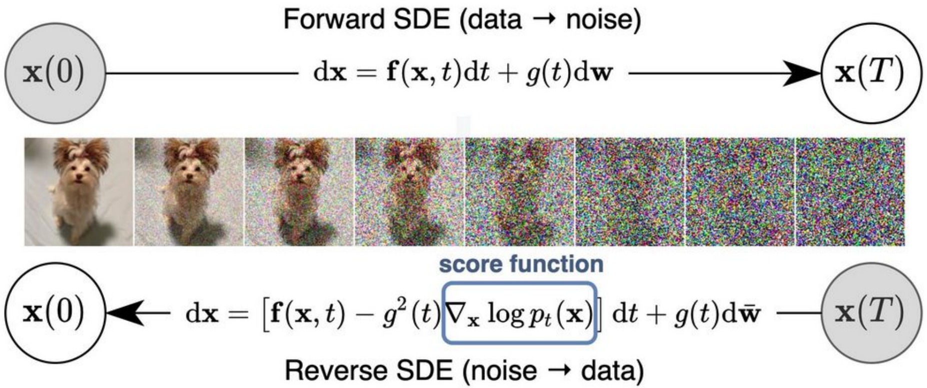
For a **Gaussian perturbation**

$$\nabla_{\tilde{\mathbf{x}}}\log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{\mathbf{x}-\tilde{\mathbf{x}}}{\sigma^2} \sim \frac{\mathcal{N}(0,1)}{\sigma}$$

- $\mathbf{s}_\theta$ is the output of the neural network
- $\boldsymbol{\lambda}(\mathbf{t})$ is a time-dependent weight function that controls the importance of each term over time

Generation

- Generation of new samples is done by solving the **reverse SDE**
- Langevin dynamics is used to draw samples from **p(x)** using only the **score function**
- High fidelity samples require small time steps,
- For Calorimeter generation, **O(100)** evaluations are enough to produce precise results

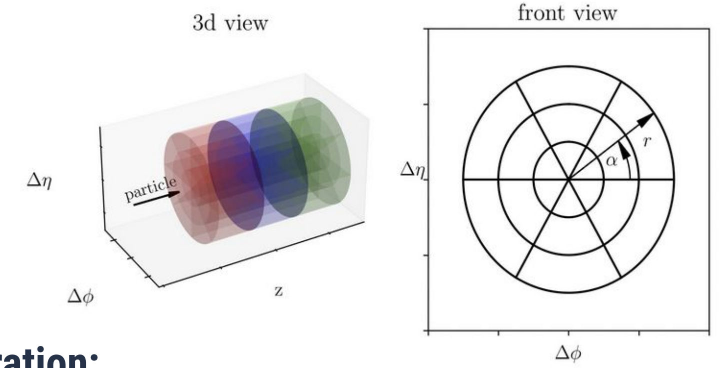


$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i, \quad i = 0, 1, \cdots, K,$$

First version of CaloScore

Let's use a realistic example: **Fast Calorimeter Simulation Challenge 2022**

- Converting initial sets voxelized in (alpha,r) coordinates to (eta,phi) coordinates
  - **Dataset 1**: 368
  - **Dataset 2**: 45x12x12 = **6480**
  - **Dataset 3**: 45x32x32 = **46080**
- **Datasets 2 and 3: 3D convolutional layers**.
  - Number of trainable parameters **~2M**
- **Dataset 1: 1D convolutional layers**
  - Number of trainable parameters **~32M**

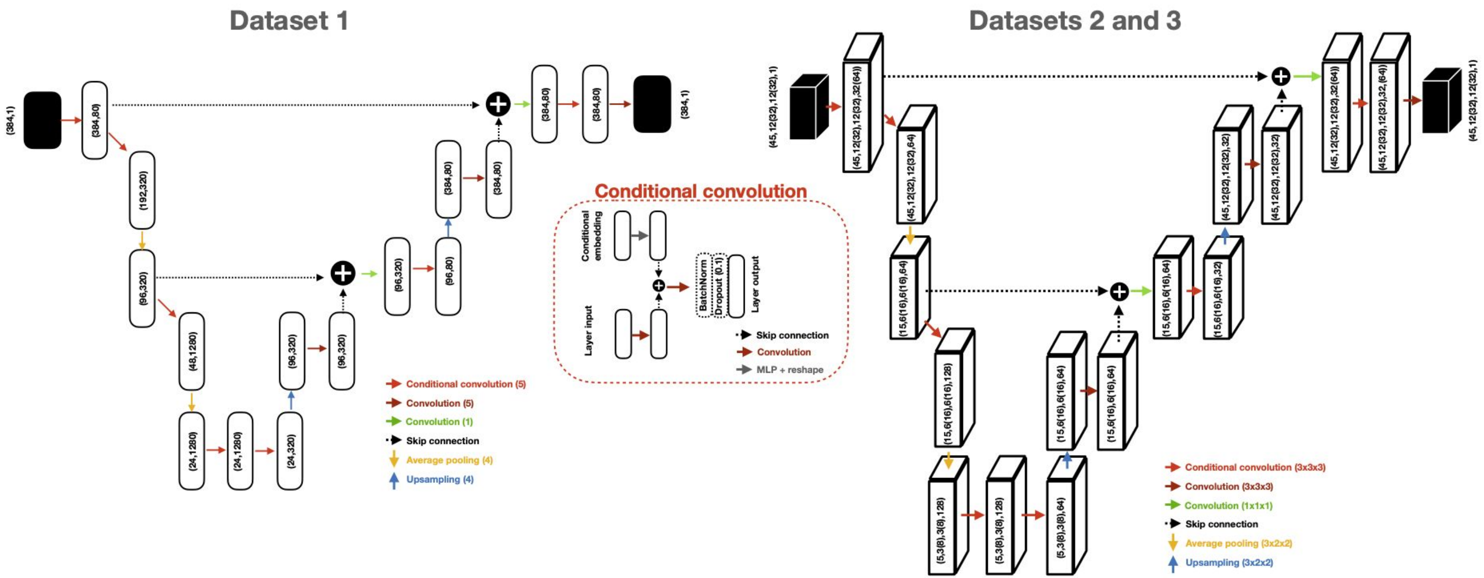


Data curation:

- Each energy deposition $\mathbf{E_i}$ is normalized by the generated energy $\mathbf{E}$ and transformed to log space: $\mathbf{u = E_i/E}$ and $u_{\text{logit}} = \log \frac{x}{1-x}$,

$$x = \alpha + (1 - 2\alpha)u \quad \text{and} \quad \alpha = 10^{-6}.$$

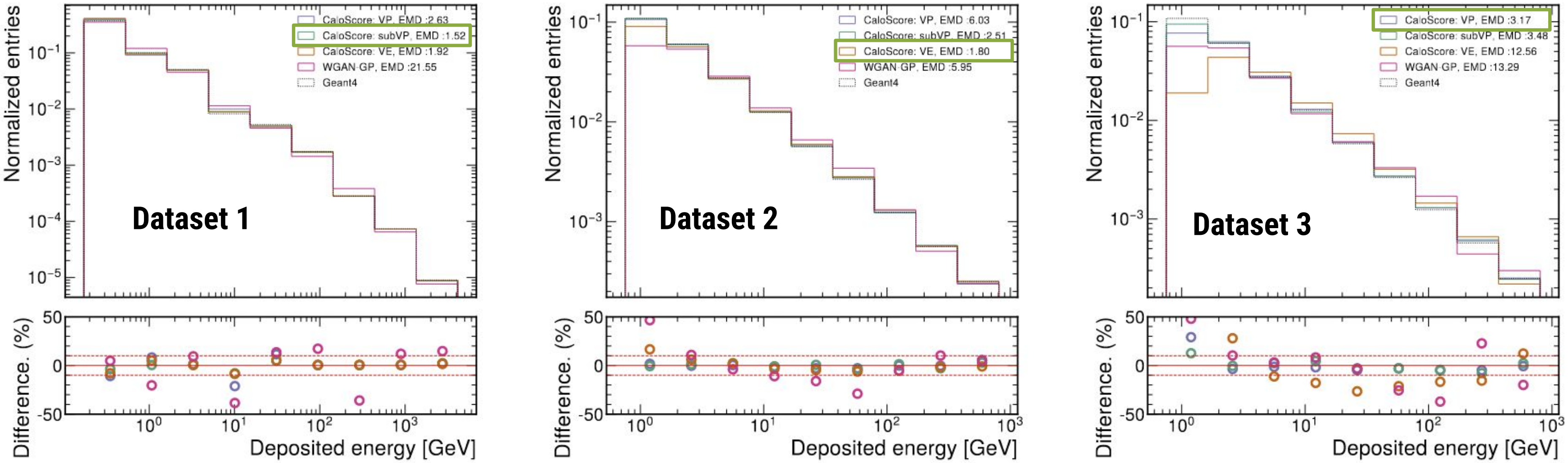
Calorimeter shower generation



Very simple **U-NET** model used to build the score function

- Lots of new developments over the years, adding attention between layers, additional skip connections, but kept it simple for this application
- **Conditional information** is added to convolutional layers as a **bias term**

Results: CaloScore v1



- **Total energy deposited in the calorimeter material**
- The 1-Wasserstein distance (**EMD**) between each generative model and Geant4 are shown for comparison

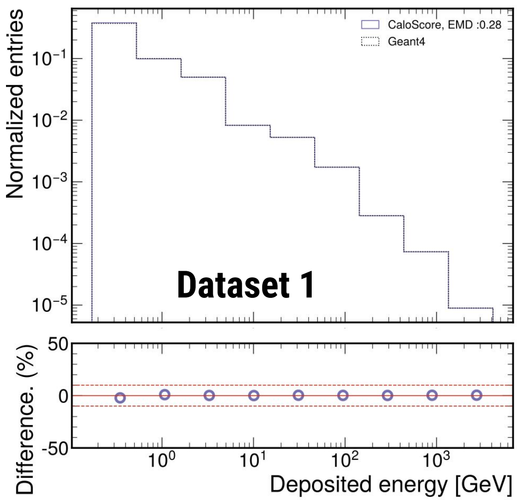
CaloScore v2

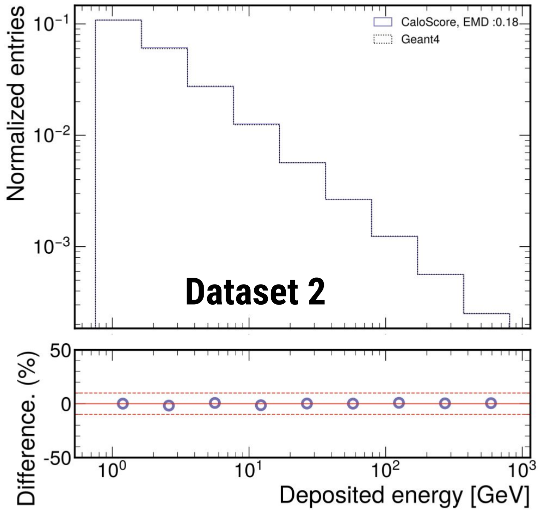
- **No additional conversion** to cartesian coordinates:
  - Use datasets 2 and 3 as is but add additional **zero-padding** to move non-empty regions closer to the center of the image
- **Replace** the basic **U-Net** backbone with **U-Net + Transformer**
  - At lower resolutions, add visual attention layers to improve the lack of inductive bias
- Break the score estimation into **2 components** trained simultaneously
  - Learn the **total energy** deposited in each layer separately from the voxel information
  - Learn only the **normalized voxels conditioned on the layer energy**
- Make the model inference faster through the use of **progressive distillation**
  - Reduce the number of steps during generation to **8** instead

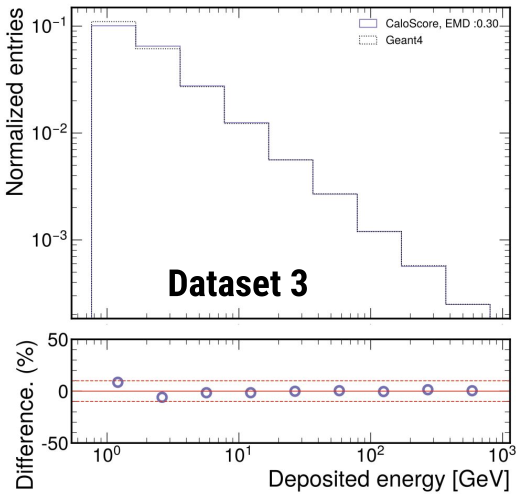
CaloScore v2

- Data curation:
  - **Normalize** between [0,1]
  - Apply **logit** transformation
  - **Standardize** with mean 0 and std 1
- Number of trainable parameters:
  - Dataset 1: **~700k**
  - Datasets 2 and 3: **~2M**
- **Learning rate schedule:**
  - Cosine Annealing with initial LR of **1e-4 * NGPUs**
- Cap **minimal energies** in the samples based on the minimum energies in the files
  - Dataset 1: **0.1 keV**
  - Datasets 2 and 3: **0.0151 MeV**

Results

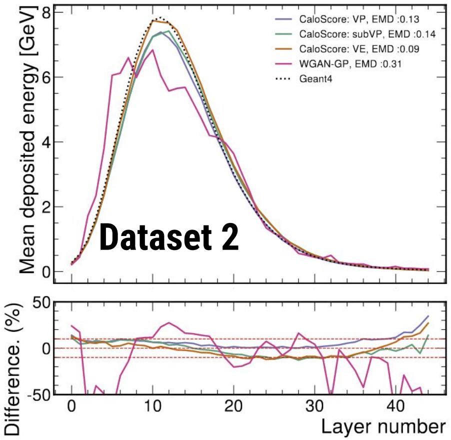


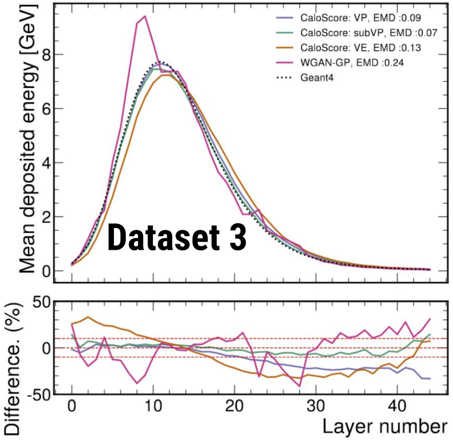


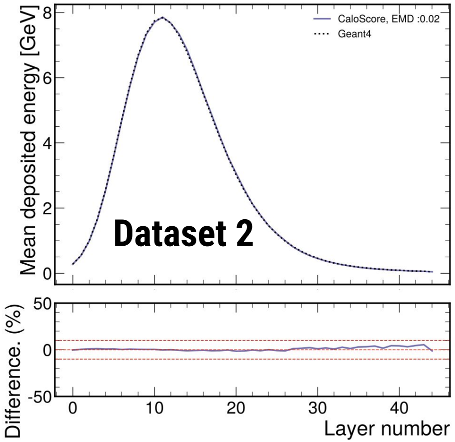


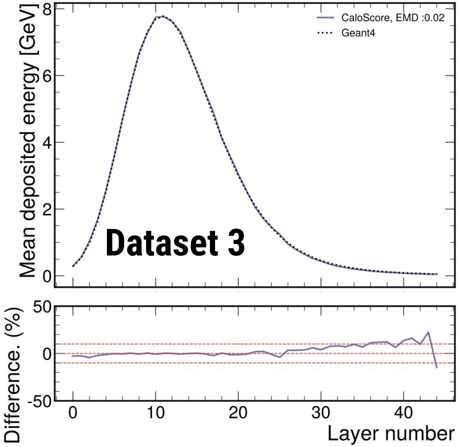
EMD	Dataset 1	Dataset 2	Dataset 3
CaloScore v1	1.52	1.8	3.17
CaloScore v2	**0.28**	**0.18**	**0.30**

Results



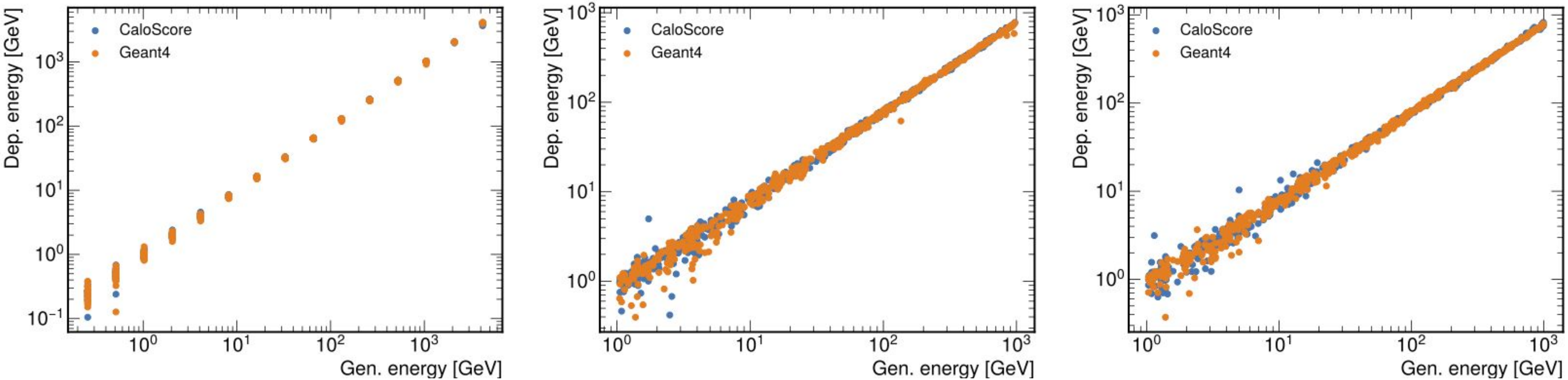






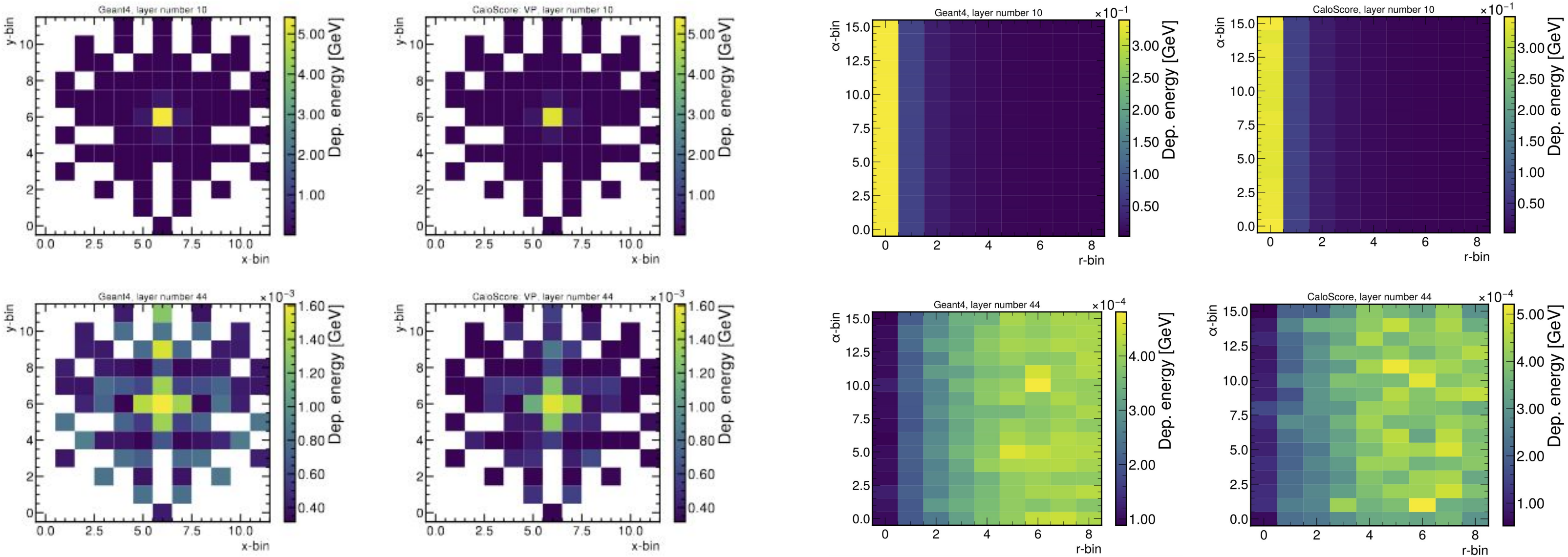
EMD	Dataset 2	Dataset 3
CaloScore v1	0.09	0.09
CaloScore v2	**0.02**	**0.02**

Results: Visualization



- **Energy spread** consistent with the simulation

Results: Visualization



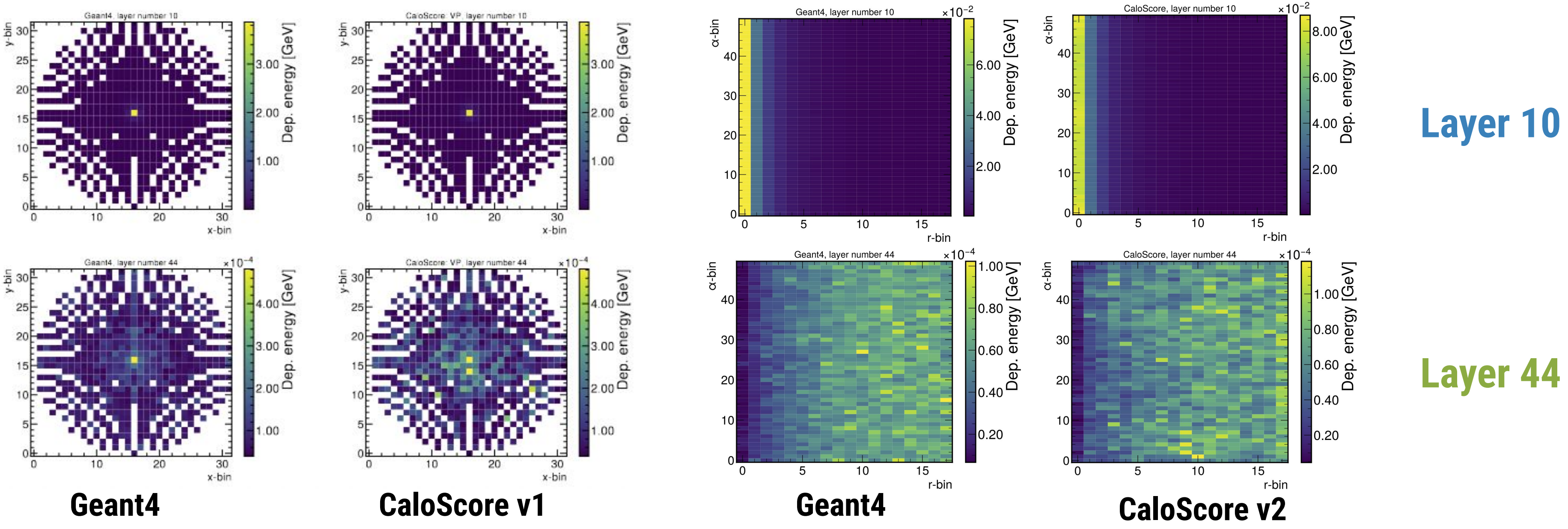
Geant4 | CaloScore v1 | Geant4 | CaloScore v2

Layer 10

Layer 44

- **Mean deposited energy** for each calorimeter layer in **dataset 2**
- Visualize the energy deposition in the layers with highest (**10**) and lowest (**44**) expected energies

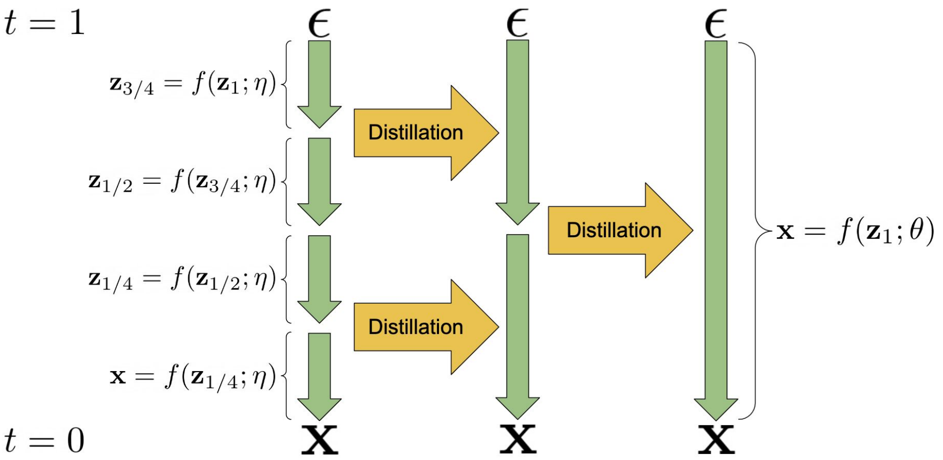
Results: Visualization



- **Mean deposited energy** for each calorimeter layer in **dataset 3**
- Visualize the energy deposition in the layers with highest (**10**) and lowest (**44**) expected energies

Results

- Progressive distillation is used to reduce the number of time steps needs during generation
- Train a follow up model that learns how to predict 2 steps at a time
- Repeat multiple times until performance degrades
- Compared to v1, the generation time is **20 times faster** for datasets 2 and 3 and **100 times faster** for dataset 1



Time to generate 100 showers [s]	Dataset 1	Dataset 2	Dataset 3
CaloScore v1	4.0	5.8	33.4
CaloScore v2	**0.034**	**0.24**	**1.47**

Conclusion

AUC/JSD **Low**	Dataset 1	Dataset 2	Dataset 3
CaloScore v2 distilled	0.9343 / 0.5324	0.7449 / 0.1446	**0.7730 / 0.1997**
CaloScore v2	**0.8513 / 0.3111**	**0.6877 / 0.0849**	-

AUC/JSD **High**	Dataset 1	Dataset 2	Dataset 3
CaloScore v2 distilled	0.6488 / 0.0781	0.8388 / 0.2854	**0.9478 / 0.5763**
CaloScore v2	**0.6266 / 0.0722**	**0.7384 / 0.1391**	-

Diffusion models are gaining popularity **inside** and **outside** HEP

- Several updates to CaloScore v1 to **address the data format** and **slow sampling times**
- Improvements on preprocessing to enforce additional energy conservation
- Excited to see how it compares against other methods!

Backup

Score matching/denoising/diffusion

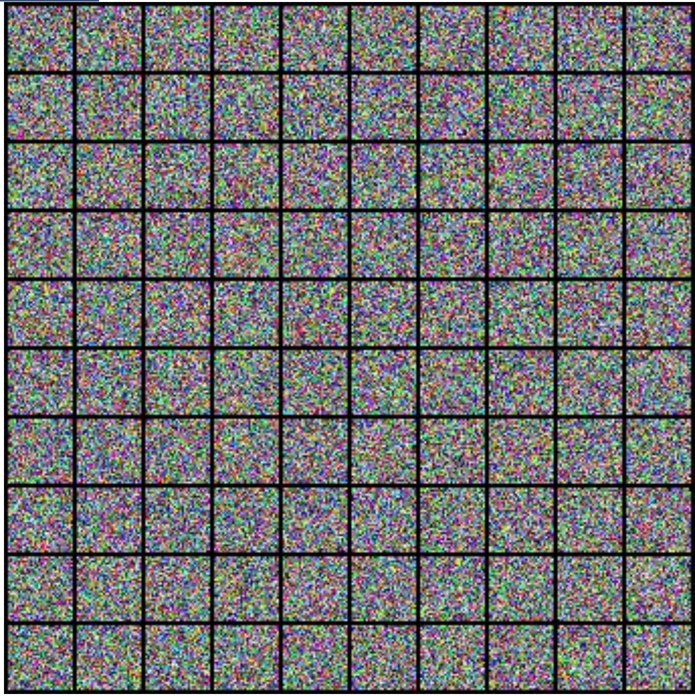
Denoise diffusion models are the newest state-of-the-art generative models for image generation.

Pros:

- **Stable training**: convex loss function
- **Scalability**: Network complexity is more sensitive to the architecture than the dimensionality
- **Access to data likelihood after training**: similar to NFs, but overall normalization is not required during training

Cons:

- **Slow sampling**: Possibly **1000s** of model evaluations to generate realistic images



Score-matching

- The common choice for **λ(t) is σ(t)²** resulting in the loss function

$$\frac{1}{2}\mathbb{E}_t\mathbb{E}_{p_t(\tilde{x})}\left[\|\sigma(t)s_\theta(\tilde{x},t)+\epsilon(0,1)\|_2^2\right]$$

- Another important result is when **λ(t) is g(t)²** that represents an [upper bound of the data likelihood]

$$\mathbf{KL}(p_0(\mathbf{x})\|p_\theta(\mathbf{x})) \leq \frac{T}{2}\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{p_t(\mathbf{x})}[\lambda(t)\|\nabla_{\mathbf{x}}\log p_t(\mathbf{x})-\mathbf{s}_\theta(\mathbf{x},t)\|_2^2] + \mathrm{KL}(p_T \parallel \pi).$$

- Allowing the **maximum-likelihood** training of diffusion models!

Likelihood estimation?

- Data generation can also be achieved by solving the **associated ODE**
  - Often leads to **worse** samples compared to Langevin dynamics generation
- On the other hand, we can also use the deterministic ODE recover the **data density!**

SDE

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

ODE

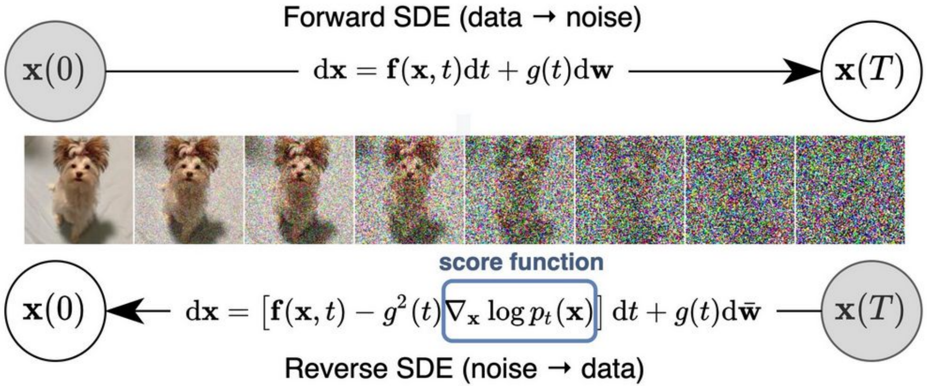
$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right]\mathrm{d}t$$

$$\mathrm{d}\mathbf{x} = \tilde{\mathbf{f}}(\mathbf{x}, t)\mathrm{d}t,$$

$$\log p_0(\mathbf{x}(0)) = \log p_T(\mathbf{x}(T)) + \int_0^T \nabla \cdot \tilde{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)\mathrm{d}t,$$

Perturbation kernels

- Let's go back to the diffusion equation
- In principle, we can choose any function for **f** and **g** but the common ones are those in which the transition kernel $p(x_t|x)$ is gaussian. That can be accomplished if **f is an affine function**



Variance preserving (VP):

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}.$$

Variance exploding (VE):

$$\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}\left[\sigma^2(t)\right]}{\mathrm{d}t}}\mathrm{d}\mathbf{w}.$$

Sub Variance preserving(subVP):

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\,\mathrm{d}t + \sqrt{\beta(t)(1 - e^{-2\int_0^t \beta(s)\mathrm{d}s})}\mathrm{d}\mathbf{w}.$$

TABLE I. Perturbation kernel induced by different SDE choices.

SDE	Perturbation kernel
VE	$\mathcal{N}(x(0), \sigma^2(t) - \sigma^2(0))$
VP	$\mathcal{N}(x(0)e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, 1 - e^{-\int_0^t \beta(s)\mathrm{d}s})$
subVP	$\mathcal{N}(x(0)e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, (1 - e^{-\int_0^t \beta(s)\mathrm{d}s})^2)$